

University of Wales Trinity Saint David

School of Engineering

Transient Thermal Diffusion Model using the Finite Difference Method

Advance Computational Methods

M6X00408

Cornelius May

1701329

26/01/2020

Contents

Problem Definition.....	3
Initial and boundary conditions	4
Mathematical Background.....	5
Finite Difference Model	5
Analysis of the code	5
Analysis	7
Limitations of the model	9
Conclusion.....	10
Bibliography	11
Appendix	12
Matlab Code.....	12
Assignment.m	12
Constants.m	12
UserInputRequest.m	13
InputChecker.m.....	14
Fourier.m.....	16
StartRecording.m	16
PlateTemperature.m.....	16
Plate0Gaussian.m	16
Plate0End2End.m.....	20
Plot.m.....	22
EndRecording.m.....	22
Other	22

Problem Definition

Within this assignment the task was to investigate the temperature propagation on a 2D plate of Aluminium of given density and conductivity. The aim was to identify the temperature at various points across the surface of the plate at time steps defined by the user.

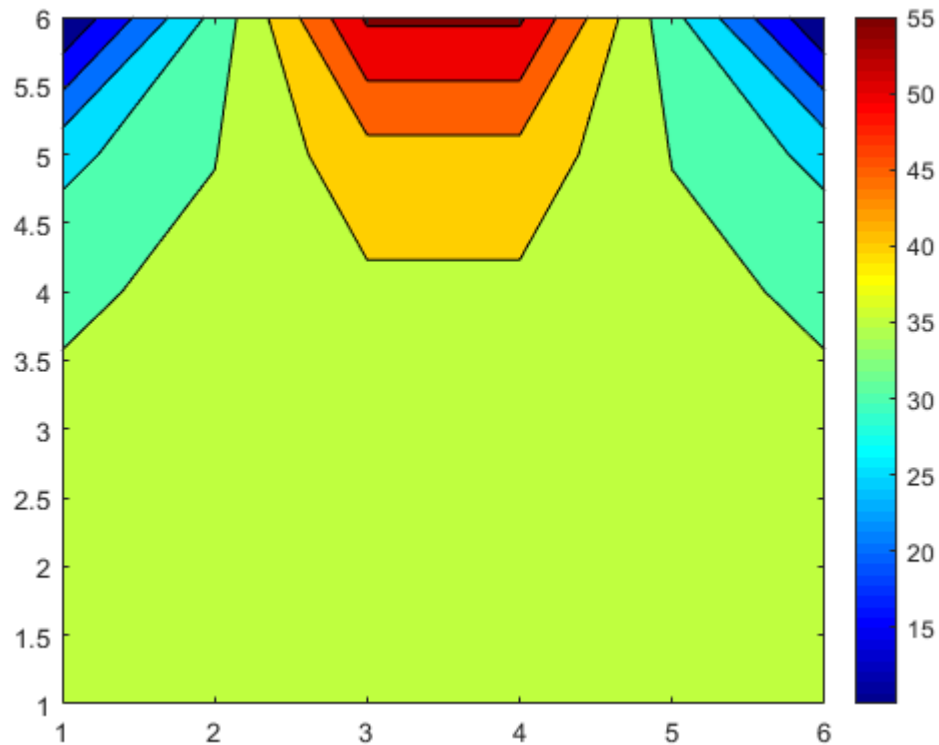


Figure 1: Example Solution

For the purpose of this investigation, the points of the compass, North, South East and West are used as directional references with point P being the local point identifying a node on the plate's surface. North refers to the cell above, East refers to the cell to the right, South to the cell below and West to the cell to the left of the cell being looked at. Each of these cells represents a node on the surface of the material.

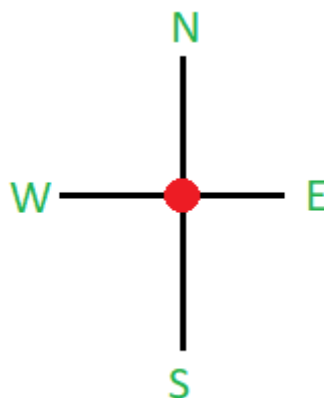


Figure 2: Locational Reference

Initial and boundary conditions

The conditions for the plate are defined in part by the user and in part by the physical properties of the material. The material properties are defined for aluminium as 142 W/m.C for Conductivity, 2800 kg/m³ for Density and 795 J/kg.C for Specific Heat (Marotin, 2019), however the plates dimensions are defined by the user.

Whilst most of the plate's initial temperature is set at 0°C , the starting temperature of the plate's north face is defined as a gaussian distribution with a temperature of 60°C at the centre and 10°C at East and West nodes. The equation used to calculate the temperature of the north face at each node can be seen in Equation 1 (*Normal Distribution*, 2020).

$$Temperature = A \times e^{\frac{-(x-x_0)^2}{2 \times SD^2}}$$

Equation 1: Gaussian Distribution

- A = Amplitude = 60°C
- $x - x_0$ = Distance between the current cell and the middle cell
- SD = Standard Deviation

$$SD = \sqrt{\frac{-(x - x_0)^2}{2 \times \ln\left(\frac{Temperature}{A}\right)}}$$

Equation 2: Standard Deviation

- Temperature = North Face Boundary Temperature = 10°C

As part of the Gaussian distribution equation, the standard deviation of the distribution is required. Rearranging Equation 1 to make the standard deviation the subject shows that the standard deviation is affected by the number of nodes from east to west on the north face of the plate being investigated. A different number of nodes would result in a different value of $x - x_0$. For this reason, it cannot remain constant for this investigation, but has to be recalculated each time the program is run and the plate's dimensions are defined. An example of what the plate might look like in its initial condition can be seen in Table 1. It is important to remember that each cell represents a node on the plate's surface.

10.0	27.1	49.2	60	49.2	27.1	10.0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Table 1: Example initial conditions

Mathematical Background

Finite Difference Model

The finite difference method was used, in order to calculate the temperature of each node of the plate at each time step. This method allows the user to approximate the result of a differential equation (*The finite difference method*, no date) with each loop gaining in accuracy and homing in on, but never quite reaching the true answer. For the purpose of this investigation, each “loop” of the finite difference model is represented by a time step, with values at time step 0 being the initial conditions and the values at each time step getting closer to the true answer.

$$\frac{d^2T}{dx^2} + \frac{d^2T}{dy^2} = \frac{1}{\alpha} \times \frac{dT}{dt}$$

- T = temperature
- t = time
- $\alpha = k/p.c$
- k = thermal conductivity
- p = density
- c = specific heat

Equation 3: Thermal diffusion equation

The Thermal Diffusion equation can be used to calculate the temperature at each node of the plate for each time step, using integration to solve the given differential equation. The issue with using this equation in its current form, is that MATLAB is not easily able to cope with second derivative equations as seen in Equation 3. For this reason, a linear equation is required.

Using the Taylor Series as seen in Equation 6 converts this differential equation to a simple linear equation that MATLAB is able to deal with. It is important to note that using the Taylor Series results in an approximation (*Taylor Series Approximation | Brilliant Math & Science Wiki*,2020), as opposed to an exact answer. The longer the simulation is left to run, the more precise the approximation becomes.

Analysis of the code

In order to make the code easy to review and debug, individual sections of the code have been separated into 11 individual ‘.m’ files. Apart from this benefit, this method of separating the code has another upside. Since individual pages can be called in any order throughout the code, instead of repeating sections of code, the page containing that section of code can be called as many times as necessary. This reduces the overall number of lines of code in the program.

The file ‘Assignment.m’ is the master index file. It calls most of the other files in the correct order. This gives a good overview of the order in which the program runs. The first file the index file calls is the ‘Constants.m’ file, which contains all the constants, such as material properties, which are not defined by the user. Next, the ‘UserInputRequest.m’ file requests the program’s user to input their desired values. Although that file does some basic checking of the values input by the user, the majority of this is done by ‘InputChecker.m’. Once the program is happy with the user’s responses the Fourier number is calculated in ‘Fourier.m’ before, if the user requested it, ‘StartRecording.m’ and ‘EndRecording.m’ convert the results of the analysis into a video file. ‘PlateTemperature.m’ decides which analysis the user has requested before, depending on the user’s request, starts either Plate0End2End.m or Plate0Gaussian.m to carry out the bulk of the analysis.

The file 'UserInputRequest.m' lets the user input values that might change from one run to the next. Decisions such as plate dimensions, grid spacing, the type of analysis to run and whether to save the results to a video file are defined by the user in this section. This file also ensures that the information input by the user does not break the Stability Criterion. Based on the grid spacing Equation 4 is used to calculate the maximum time step permitted to be used by the user. If this criterion is not met, the user is informed and given the option to use the suggested time step, or to insert a valid value.

$$Stability\ Criterion = \frac{GridSpacing^2}{4 \times \alpha}$$

Equation 4: Stability Criterion

The majority of the input checking is done by 'InputChecker.m'. Here the program first ensures that the user has not left any inputs blank, before ensuring the input values are valid. It is important to make sure that the grid spacing is less than the x and y dimensions and that it is a common denominator of both of these values. At each stage, if there are any values that do not meet the set criteria, the user is informed of the problem. They are then given the chance to try again, by inputting a value that is allowed. Alternatively, the user can use a value recommended by the program. The program does this by using the code in Figure 3 to find a common denominator between the x and y lengths.

```
SuggestedGridspacing = gcd(XAxisLength,YAxisLength); % Calculate a suggested grid spacing
```

Figure 3: Determine Lowest Common Denominator

Once the user has input valid values, the program can autonomously resolve the remainder of the analysis. The syntax "tic" starts a timer at this point allowing the user, upon completion of the program, to see how long the program needed to complete the analysis. The syntax "toc", at the end of the program, tells the timer that the analysis has finished.

For a cleaner program, made up of fewer files, it could be suggested that the Fourier number should be calculated in the 'Constants.m' file as opposed to the 'Fourier.m' file. The issue with this is that the grid spacing, input by the user, is required before it can be calculated. This is why it has been put into its own file.

The 'StartRecording.m' file initialises the video file to which the output from the analysis can be saved. This only occurs if the user has requested this type of output during the input stage. Settings for the video file, such as the file's name and frame rate are defined here. Multiple other settings could also be defined at this point however, since the default values are acceptable, they have not been specifically defined. Within the Plate0End2End.m or Plate0Gaussian.m files, figures are created and saved to the video file. Although saving every graph created is possible, this increases the time required to calculate a result. This is why only every 10th graph is saved. In 'EndRecording.m' the video file is closed and saved for later playback.

If the user has selected the End Heated Plate analysis 'PlateTemperature.m' will finally run the file called 'Plate0End2End.m'. This file does the majority of the work in analysing the temperature changes at each node of the plate. After working out how many rows and columns are required based on the dimensions of the plate, the program initialises a matrix and fills each cell with values of zero. This prevents issues with matrix dimensions later. Now the plate temperatures at time 0 are input into the matrix that was just created. For this analysis, the east and west faces of the plate, now represented by the first and last column of the matrix, are already known and can be input for

every row. Then the rest of the values in the first and last row can be calculated, as these are a linear distribution between the first and last column. At this point the rest of the plate remains at 0. Now at each time step the equation in **Error! Reference source not found.** is used to find the change in temperature for the remaining cells. At the end of each time step, a graph is created using a colour code to display the relative temperature in each cell.

```
Plate (RowCounter, ColumnCounter) = Fourier0 * (OldPlate (RowCounter, ColumnCounter + 1) + OldPlate (RowCounter, ColumnCounter - 1) + OldPlate (RowCounter + 1, ColumnCounter) + OldPlate (RowCounter - 1, ColumnCounter)) - (4 - (1 / Fourier0)) * Tp);
```

Equation 5: Temperature of cell at time dt

If the gaussian distribution analysis was chosen by the user, the Plate0Gaussian.m file is run. Unlike in the previous analysis, after calculating the number of columns and rows and creating a matrix of 0s, this file only defines the first row of the matrix for time 0. Since the values in this row are distributed along a gaussian curve, first the standard deviation needs to be evaluated for the number of columns in the given matrix. This is then used to calculate the remaining values in the first row. Unlike with the linearly distributed plate, the remaining three sides of the plate are not constant for this investigation. As a result, a similar equation as the one in **Error! Reference source not found.** is used. Upon looking at this equation, an issue becomes apparent as this equation relies on there being further cells on all four sides of the cell being calculated. Since on the boundary of the plate this is not the case, any temperatures that are missing are assumed to be parallel to the other side of the cell. This is as a result of these boundaries being perfectly insulated and as a result, in perfect equilibrium. Finally, the equation in **Error! Reference source not found.** is used to calculate the temperature in the remaining cells at each time step. As before, a graph is created at each time step to display the change in relative temperature for the user.

The last thing that has to be decided by the program is when to end. Until a given exit criteria is met, the program loops, calculating the change in temperature of each cell at each time step. In order to determine when to stop, the program compares the difference between the previous value and the newly calculated value of a given cell. If the change in a cell's value is determined to be so small the it can be considered negligible, a 1 is placed in the corresponding position of a matrix with the same dimensions as the plate temperature matrix. Once the temperature in each cell has been calculated for a time step, the number of 1s in this matrix is counted. The program then moves on to the next time step, until it has determined that there is a 1 in every cell of the matrix, at which point the change in every cell is negligible. This is then determined as the end of the program.

Analysis

Looking at the results of the analysis, it quickly becomes apparent that some level of user understanding is required. The user's input can have a large impact on the precision of the results. On the other hand, even with reduced precision, their accuracy will not change, irrespective of what the user inputs. As long as the plate's X and Y dimensions are correct for what the user is investigating, the other values simply improve on precision.

Although the analysis was intended to be carried out on a plate with a gaussian distribution on the north face and insulation on the other three faces, these results are difficult to confirm as it is unclear what temperature would be expected on each node. For this reason, a similar analysis was coded looking at a end heated plate, with known boundary conditions, expected to result in a plate with vertical rows of similar values linearly distributed across the horizontal axis. This would allow the evaluation of the equation used to calculate the temperature at each node for each time step.

The end heated plate used for analysis of the code has a starting temperature of 50°C on the west face, 0°C on the east face, whilst the temperature of the north and south face temperatures are linearly distributed between the two. All other temperatures are 0°C at time 0.

Figure 4 helps to demonstrate the importance of the users input. Both plates are analysing results for the plate of the same dimensions once it has reached its equilibrium point. The first thing to note is that despite the differences between the plates, they both show the same temperature distribution, with the extremes of hot and cold in the same areas. Furthermore, they both show the vertical and horizontal temperature relationships expected, confirming with some level of confidence, that the code and equations being used are giving accurate results.

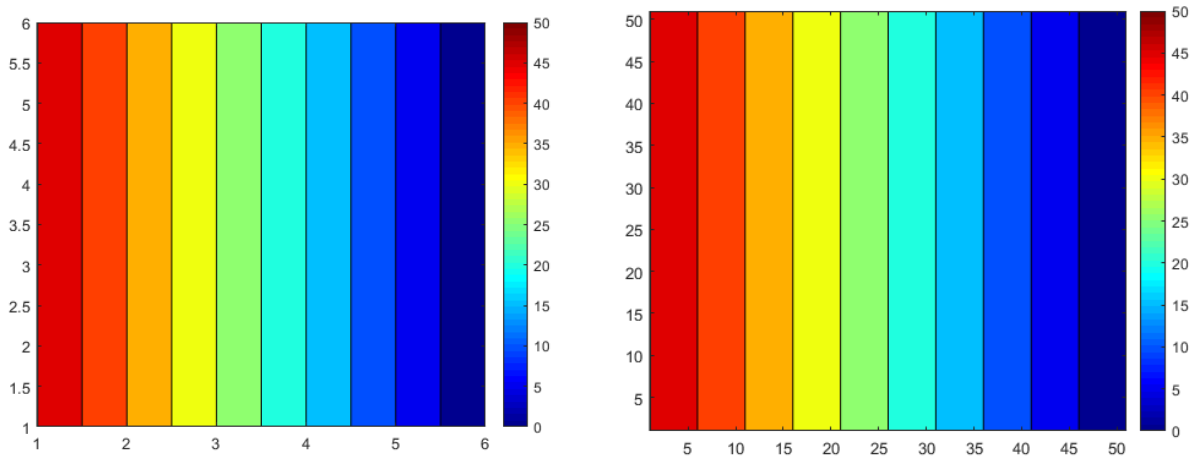


Figure 4: 5 x 5 x 1m at 30s (Left) and 5 x 5 x 0.1m at 30s (Right)

The first area, where the user's input must be carefully considered is the grid spacing. In both investigations, a time step of 30 seconds was used, however due to the increased number of nodes, the more detailed investigation took 149.2 seconds while the less detailed investigation took only 105.4 seconds. Furthermore, the programs prediction of the time taken to reach equilibrium is less accurate. Whilst the 1-meter grid spacing was predicted to take 58.5 hours to reach equilibrium, the 0.1-meter grid spacing plate, with the exact same dimensions and time step took only 57.8 hours to reach equilibrium. The user must determine what level of accuracy is required. This might not only be determined by the accuracy of the results themselves, but also by the time available to complete the analysis.

Another area of importance is the time step. This is the interval at which plate temperatures are investigated. Whilst a lower value time step will allow the program to more accurately predict the time required to reach equilibrium, this will also result in the analysis taking longer to complete, as more steps have to be calculated between the start and equilibrium states. For this reason, depending on the grid spacing chosen by the user, the program advises the user what time step to use, and will only allow a time step that will result in the Fourier's number being less than 0.25. As with the grid spacing, the user can determine what level of accuracy they require when choosing the time step. Furthermore, to make comparison between different runs easier, the user might want to consider using the same time step from one run to the next.

The final area to consider is the level of accuracy used for the exit criteria. The value of each cell is compared to the value in the same location at the previous time step. If the difference between the two values is small enough, the cell is considered to be in equilibrium and once all cells meet the exit criteria, the entire plate is in equilibrium. The ideal value of similarity was determined to be 99.99999%. Comparing Figure 5 and Figure 4 shows the effect of reducing this level of accuracy. In

Figure 5 the similarity from time step to time step was reduced to only 99.9% and as a result it is possible to see that obviously thermal equilibrium has not been reached, as the temperature strips would be expected to be vertical. On the other hand, the investigation was carried out with an accuracy of 99.999999%, resulting in the time taken for a solution to be reached being significantly increased. The value chosen was found to be an acceptable balance between accuracy and analysis time (Merziger *et al.*, 2014).

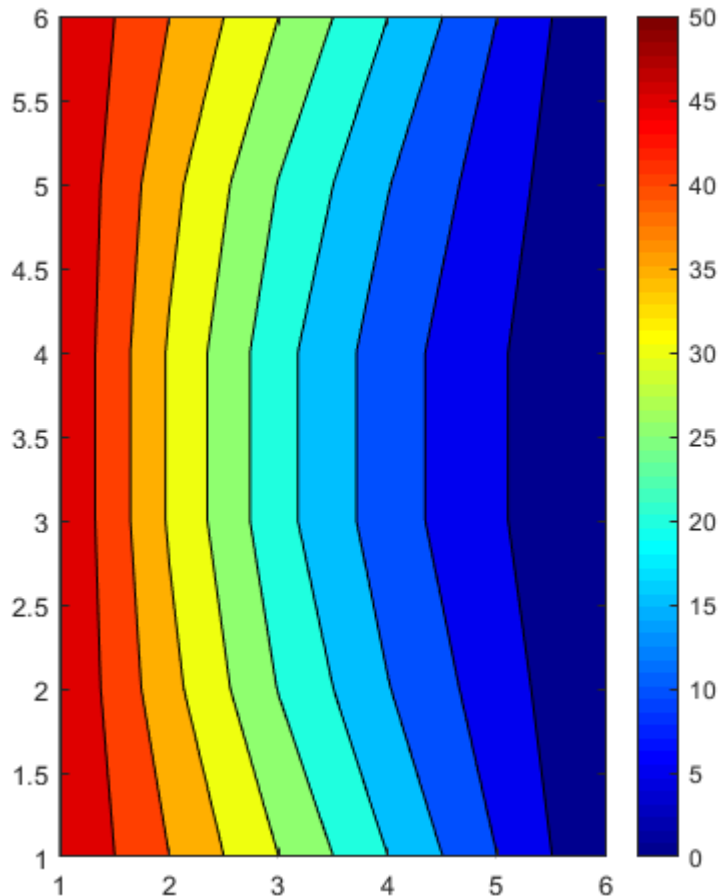


Figure 5: 5 x 5 x 1m at 30s with 99.9% similarity

Limitations of the model

The program has been written to analyse a very specific set of circumstances, meaning that it is not designed in a way that would allow more abstract shapes or significantly different boundary conditions. It is restricted to analysing only a rectangular shaped material with a predetermined thickness. Further work could involve the option for the user to input a wider range of starting parameters. This might include the option to input a starting temperature for each node and the possibility to define a material thickness. Furthermore, the ability to analyse more abstract shapes could further improve the usefulness of the program.

Giving the user the option to edit the material properties, either by offering a list of materials to choose from or by allowing the user to input individual properties manually, would further develop the program, giving it further flexibility in its application.

Another limitation of the model is the number of decimal places MATLAB has available to each variable, as maximum of 30 integers can be used per variable. This limit becomes significant when attempting to determine the time step at which thermal equilibrium is reached. The program currently defines thermal equilibrium as the point there the temperature of a cell changes by less than 0.00000001 degrees Celsius per time step. Even with the limitation set by MATLAB a higher degree of accuracy is possible within the exit criteria however, depending on the time step and grid spacing used, this higher accuracy was not determined to be necessary. A piece of potential further work could be to allow the user to determine the level of accuracy required, giving them the flexibility to increase or decrease this value, depending on requirements and time available.

A further limitation is down to the equation used to calculate the temperature in each cell. Results can only be trusted when the grid spacing (h) and time step (s) carry a certain level of similarity. For this reason, the user of the program is only allowed to use values for the time step that will result in the Fourier's number falling between 0.25. This is dependent on the grid spacing chosen by the user.

Conclusion

The program is a good way to gain a better understanding of how heat propagates through the plate. It allows the user to carry out some basic level of analysis with confidence in their accuracy and with the precision of the results being guaranteed.

As a result of the analysis carried out on the linearly distributed plate, there is a significant level of confidence that the Gaussian distribution plate created will show accurate results. Although the initial conditions between the two plates are different, the equations used to calculate the change in temperature for the individual cells are the same, so since there is evidence to show that these perform as we would expect on the linearly distributed plate, there is no reason to suspect them to be incorrect for other applications.

A small amount of future work could greatly improve upon the program with the groundwork already being in place for a more flexible and diverse version. With the option of investigating a wider range of 2D shapes, potentially of different materials and less well-defined initial temperatures, the program might be able to find more applications in real world testing.

Finally, real world testing could conclusively confirm the accuracy of the program. A rectangular plate of known dimensions could be heated to the same initial starting conditions as in the program and insulation used on the three none heated edges. Using temperature probes distributed across the surface in the location of the nodes, the temperature change could be monitored and compared to the results of the program. This might give some confidence of the program's accuracy.

Bibliography

Marotin, A. (2019) *Transient Thermal Diffusion Model using the Finite Difference Method - Brief*. Swansea.

Merziger, G. et al. (2014) *Formeln + Hilfen Höhere Mathematik*. Barsinghausen: Binomi Verlag.

Normal Distribution (no date). Available at: <https://stattrek.com/probability-distributions/normal.aspx> (Accessed: 24 January 2020).

Taylor Series Approximation | *Brilliant Math & Science Wiki* (no date). Available at: <https://brilliant.org/wiki/taylor-series-approximation/> (Accessed: 24 January 2020).

The finite difference method (no date).

Appendix

Matlab Code

Assignment.m

```
clc
clear

%% This Document runs all subsequent .m files

Constants % All constants are defined within this document

UserInputRequest % Calls scrips UserInputRequest.m which requests user to
input values

InputChecker % Check that values input by the user meet criteria and will
not cause error in the program

tic % Begin counting time required for entire program to run

Fourier % Calculate Fourier's Number

if VideoCheck == 1 % Check if the user would like the output to be recorded
as a video
    StartRecording
end

PlateTemperature % Calculate plate temperature

if VideoCheck == 1
    EndRecording % Output video file
end

toc % End counting time required for entire program to run
disp(strcat('Time taken to thermal equilibrium: ',num2str(TimePassed),'s'))
%Display the time taken to reach thermal equilibrium
```

Constants.m

```
%% Known Constants

k = 143; %Conductivity of aluminium (W/m.C)
p = 2800; %Density of aluminium (kg/m^3)
c = 795; %Specific Heat Capacity of aluminium (J/kg.C)
alpha = k / (p*c);

%% Part 1: Un-Insulated with 50degC on the left and 0degC on the right
(End2End)

TempL = 50; %West surface temperature
TempR = 0; %East surface temperature
```

```
%% Part 2: Insulated on 3 sides with a single point head source (Gaussian)
```

```
Amplitude = 60; %Maximum temperature on the gaussian distribution plate  
BoundaryTemp = 10; %Temperature at the edge of the north face of the  
gaussian distribution plate  
SourceTemp = Amplitude; %Re-naming variable for consistency with the norm.
```

UserInputRequest.m

```
%%User input
```

```
VideoCheck = menu('Would you like to record the results as a video file?  
This will take longer!', 'Yes Please', 'No Thank you'); %Ask the user if they  
would like the plate figures to be output as a video file
```

```
Test = menu('Would you like to evaluate an insulated plate, or an end to  
end heated plate?', 'Insulated Plate (Gaussian)', 'End Heated Plate');  
%Determine which scenario is being used.
```

```
if Test == 1 %Checking which test would like to be run  
    Test = ('I'); %Gaussian  
elseif Test == 2  
    Test = ('E'); %End heated  
else %Invalid response  
    disp('That wasnt an option. The Gaussian plate will be calculated.')
```

```
    Test = ('I'); %Gaussian is automatically run
```

```
end
```

```
prompt = {'Enter plate dimension (/m) - x:', 'Enter plate dimension (/m) -  
y:', 'Enter grid spacing (/m)'}; %Defining the values the are required  
dimentions = inputdlg(prompt); %Pop-up dialog box for the user to complete  
XAxisLength = str2num(cell2mat(dimentions(1,1))); %Splitting up results  
into variables for later use  
YAxisLength = str2num(cell2mat(dimentions(2,1)));  
GridSpacing = str2num(cell2mat(dimentions(3,1)));
```

```
clear prompt %No longer needed so clearing for potential later use and to  
keep the workspace clean  
clear dimentions
```

```
%% Ensure stability criterion cannot be broken.
```

```
StabilityCriterion = ((GridSpacing^2)/(4*alpha)); %What is the maximum time  
step that can be used?
```

```
disp('Thankyou. Now, please define the time step in seconds. The maximum  
permissable =') % Informing the user of the maximum time step they are  
allowed to request
```

```
disp(StabilityCriterion)
```

```
TimeStep=input('Time Step(s) - '); %Requesting definition by user.
```

```
if TimeStep>StabilityCriterion %Is user input permitted?  
    disp('Based on the grid spacing you have defined, this time step is not  
permitted. The maximum permitted time step =')  
    disp(StabilityCriterion)  
    disp('This stability criterion will now be used')  
    TimeStep = StabilityCriterion; %Overrulling user input to use maximum  
permitted value.
```

end

InputChecker.m

```
%% Ensure Values are not Zero
```

```
counter = 1;
```

```
while counter < 3 %The user has this many tries to get it right.
```

```
    if isempty(XAxisLength) == 1 || isempty(YAxisLength) == 1 ||  
    isempty(GridSpacing) == 1 %Did the user leave any of these values blanc?
```

```
        disp('You have entered one or more blanc values. The program will  
restart. Please input real numbers.')
```

```
        UserInputRequest %Re-run the script requesting user inputs.
```

```
        counter = counter + 1; %The user has used up one of their tries.
```

```
    else
```

```
        counter = 4; %The user got it right so does not need any more  
tries.
```

```
    end
```

```
end
```

```
clear counter %For use later and to keep the workspace tidy
```

```
if isempty(XAxisLength) == 1 || isempty(YAxisLength) == 1 ||  
isempty(GridSpacing) == 1 %The user has used up all their attempts to input  
values. One final check.
```

```
    disp('Even after all your attempts you have still left a value blanc.  
The program is going to crash now, thanks...') %Informing the user why the  
program is crashing.
```

```
end
```

```
%% Ensure the spacing complies with the dimensions of the plate
```

```
counter = 3; %Three attempts
```

```
InPutCounter = 1; %How many input attempts have they had?
```

```
while InPutCounter<counter %Check that the user has not used up all of  
their attempts.
```

```
    if XAxisLength>=GridSpacing && YAxisLength>=GridSpacing %Check that  
grid spacing is smaller than plate dimensions. Check Stability critaria are  
met.
```

```
        InPutCounter = 4; %The values are acceptable, so no further  
attempts are required.
```

```
    elseif XAxisLength<GridSpacing || YAxisLength<GridSpacing
```

```
        disp('Sorry. Your grid spacing appears to be too big, please choose  
a value smaller than both your X and Y axis length and try again.')
```

```
%Guiding the user toward a valid input.
```

```
        UserInputRequest %Re-run the script requesting user input.
```

```
        InPutCounter=InPutCounter+1; %One of the attempts has been used up.
```

```

end
end

%% Ensure grid spacing is a factor of plate dimensions

checkx = ~rem(XAxisLength,GridSpacing); %Ensure that the GridSpacing is
valid compared to the XAxisLength.
checky = ~rem(YAxisLength,GridSpacing); %rem ensures values values are
factors.

counter = 4; %Three attempts
InPutCounter = 1; %How many input attempts have they had?

while InPutCounter<counter %Check that the user has not used up all of
their attempts to get it right.

    if checkx == 0 || checky == 0 %Gridspacing is not a factor of x or y
dimensions.
        SuggestedGridspacing = gcd(XAxisLength,YAxisLength); % Calculate a
suggested grid spacing
        disp(SuggestedGridspacing) %Show the user the suggested brid
spacing.
        decision = input('Your grid spacing is not going to work. Would you
like to use the value above? (y/n)','s'); %Give the user the opportunity to
use the suggested grid spacing.

        if decision == ('y') %Has the user decided to go with the given
advice?
            InPutCounter = 4; %No further attempts needed.
            GridSpacing = SuggestedGridspacing; %Use the suggested grid
spacing
        elseif decision == ('yes') %Despite attempts to show the user how
to input the decision they might spell out their decision.
            InPutCounter = 4; %No further attempts needed
            GridSpacing = SuggestedGridspacing; %Use the suggested grid
spacing
        elseif InPutCounter == 3 %The user has gone against advice too
often.
            disp('Too many attempts. Grid spacing will be set to suggested
value.')
            GridSpacing = SuggestedGridspacing; %Use the suggested grid
spacing
            InputCounter = InPutCounter +1; %One attempt has been used up
        else %Either the user does not want to use the value, or their
input is not recognised.
            InPutCounter = InPutCounter + 1; %One attempt has been used up
            disp('Please try again.')
            UserInputRequest %Re-Offer the user input questions
        end
    else %Grid spacing is valid.
        InPutCounter = 4; %No further attempts required
    end

end

clear counter %Clean up the workspace
clear InPutCounter %Clean up the workspace

```

Fourier.m

```
%% Calculate Fourier's Number

Fourier0 = (alpha * TimeStep)/(GridSpacing^2); %Calculated using constants
and values calculated from user input.
```

StartRecording.m

```
writerObj = VideoWriter('myVideo.avi'); %Format and name of the video file
writerObj.FrameRate = 1; %Frame rate of the video file

open(writerObj); % open the video writer
```

PlateTemperature.m

```
%% Which plate has the user decided to calculate.

if Test == ('E') %The linear distribution end heated plate has been chosen
    Plate0End2End
elseif Test ==('I') %The insulated gaussian distribution heated plate has
been chosen
    Plate0Gaussian
end
```

Plate0Gaussian.m

```
%% Initial Plate Temperature (Gaussian)

%% PlateGaus

%% Create Matrix of Temperatures at T=0

NumberOfColumns = (XAxisLength/GridSpacing)+1; %Calculate the number of
columns based on the user's defined dimentions
NumberOfRows = (YAxisLength/GridSpacing)+1; %Calculate the number of rows
based on the user's defined dimentions

if rem(NumberOfColumns,2) ~= 0 %Odd number of Columns

    SD = sqrt((- (NumberOfColumns-
((NumberOfColumns+1)/2))^2)/(log(BoundaryTemp/SourceTemp)*2)); %Calculate
standard deviation of gaussian distribution

    Plate = zeros(NumberOfRows,NumberOfColumns); %Create matrix of plate
dimentions filled with zeros

    Plate(1,((NumberOfColumns+1)/2)) = SourceTemp; %Define the temperature
in the middle of the plate

    StepNum = (NumberOfColumns-3)/2; %How many cells need to be populated
```

```

%      TempStep = (SourceTemp-BoundaryTemp)/((NumberOfColumns-3)/2);

    counter = 1;
    while counter<=NumberOfColumns
        Plate(1,counter) = Amplitude*exp(-(counter-
((NumberOfColumns+1)/2))^2)/(2*(SD^2))); %Calculate temperature of all
cells in the first tow

        counter = counter + 1;
    end

else %Even number of columns

    SD = sqrt(-(NumberOfColumns-
((NumberOfColumns+1)/2))^2)/(log(BoundaryTemp/SourceTemp)*2)); %Calculate
standard deviation of gaussian distribution

    Plate = zeros(NumberOfRows,NumberOfColumns); %Create matrix of plate
dimentions filled with zeros

    %      Plate(1,((NumberOfColumns)/2)) = SourceTemp; %Define the
temperature in the middle of the plate
    %      Plate(1,((NumberOfColumns)/2)+1)) = SourceTemp; %Define the
temperature in the middle of the plate

    StepNum = (NumberOfColumns-2)/2; %How many cells need to be populated

%      TempStep = (SourceTemp-BoundaryTemp)/((NumberOfColumns-2)/2);

    counter = 1;
    while counter<=NumberOfColumns/2
        Plate(1,counter) = Amplitude*exp(-(counter-
((NumberOfColumns+1)/2))^2)/(2*(SD^2))); %Calculate temperature of the
first half of cells in the first row
        Plate(1,(NumberOfColumns+1)-counter) = Amplitude*exp(-(counter-
((NumberOfColumns+1)/2))^2)/(2*(SD^2))); %Calculate temperature of teh
second half of cells in the first row, symetrically to the first half

        counter = counter + 1;
    end

end

Plate0 = Plate; %Save as initial plate

%% Update temperatures at T=dt

TempCheck = 0;
PlateCounter = 1;
ExitHold = zeros(NumberOfRows,NumberOfColumns);
ExitHold(1,:) = 1;
ExitHold(NumberOfRows,:) = 1;
ExitHold(:,1) = 1;
ExitHold(:,NumberOfColumns) = 1;

```

```

ExitCriteria = NumberOfRows * NumberOfColumns;
ExitSum = 0;
PlotCounter = 1;
TimeCounter = 0;

while ExitSum < ExitCriteria

    OldPlate = Plate; %Save plate from last time step

    ColumnCounter=2; %Start in the second column

    while ColumnCounter<NumberOfColumns
        RowCounter =2; %Start in second row
        while RowCounter<NumberOfRows
            Tp = Plate(RowCounter,ColumnCounter); %Saves the previous value
of a cell

            Plate(RowCounter,ColumnCounter)=Fourier0*(OldPlate(RowCounter,ColumnCounter
+1)+OldPlate(RowCounter,ColumnCounter-
1)+OldPlate(RowCounter+1,ColumnCounter)+OldPlate(RowCounter-
1,ColumnCounter)-(4-(1/Fourier0))*Tp); %Calculates new value in this cell

            ExitCheck = Tp/Plate(RowCounter,ColumnCounter); %How much has
the value changed?

            if ExitCheck > 0.99999999 %Has the value changed enough to meet
the exit criteria?
                ExitHold(RowCounter,ColumnCounter) = 1;
            %
            elseif
Plate(RowCounter,ColumnCounter)==Plate(RowCounter,ColumnCounter+1)
            %
                ExitHold(RowCounter,ColumnCounter) = 1;
            end

            RowCounter = RowCounter + 1;

        end
        ColumnCounter = ColumnCounter + 1;
    end

    RowCounter = 2;

    while RowCounter<NumberOfRows
        Tp = Plate(RowCounter,1);

        Plate(RowCounter,1)=Fourier0*(OldPlate(RowCounter,1+1)+OldPlate(RowCounter,
1+1)+OldPlate(RowCounter+1,1)+OldPlate(RowCounter-1,1)-(4-
(1/Fourier0))*Tp); %Calculates new value in the first column
        Tp = Plate(RowCounter,NumberOfColumns);

        Plate(RowCounter,NumberOfColumns)=Fourier0*(OldPlate(RowCounter,NumberOfCol
umns-1)+OldPlate(RowCounter,NumberOfColumns-
1)+OldPlate(RowCounter+1,NumberOfColumns)+OldPlate(RowCounter-
1,NumberOfColumns)-(4-(1/Fourier0))*Tp); %Calculates new value in the last
column

        RowCounter = RowCounter +1;
    end

    ColumnCounter = 2;

```

```

while ColumnCounter<NumberOfColumns
    Tp = Plate(NumberOfRows,ColumnCounter);
    Plate(NumberOfRows,ColumnCounter)=Fourier0*(OldPlate(NumberOfRows-
1,ColumnCounter)+OldPlate(NumberOfRows-
1,ColumnCounter)+OldPlate(NumberOfRows,ColumnCounter+1)+OldPlate(NumberOfRo
ws,ColumnCounter-1)-(4-(1/Fourier0))*Tp); %Calculates new value in the last
row

    ColumnCounter = ColumnCounter +1;
end

    Tp = Plate(NumberOfRows,1);
    Plate(NumberOfRows,1)=Fourier0*(OldPlate(NumberOfRows-
1,1)+OldPlate(NumberOfRows-
1,1)+OldPlate(NumberOfRows,1+1)+OldPlate(NumberOfRows,1+1)-(4-
(1/Fourier0))*Tp); %Calculates new value in the last row
    Tp = Plate(NumberOfRows,NumberOfColumns);
    Plate(NumberOfRows,NumberOfColumns)=Fourier0*(OldPlate(NumberOfRows-
1,NumberOfColumns)+OldPlate(NumberOfRows-
1,NumberOfColumns)+OldPlate(NumberOfRows,NumberOfColumns-
1)+OldPlate(NumberOfRows,NumberOfColumns-1)-(4-(1/Fourier0))*Tp);
%Calculates new value in the last row

%     PlateName = strcat('Plate',num2str(PlateCounter));
%     variable.(PlateName) = Plate ;

XAxis = zeros(NumberOfColumns,1);
counter = 1;

while counter<=NumberOfColumns
    XAxis(counter,1) = counter;
    counter = counter+1;
end

YAxis = zeros(1,NumberOfRows);
counter = 1;

while counter <=NumberOfRows
    YAxis(1,counter) = counter;
    counter = counter+1;
end

%% Plot Figure

if PlateCounter > PlotCounter +10 || PlotCounter == 1 %Plot table to
figure on the first and every subsequent 10 plates calculated.

    Plot

    if VideoCheck == 1
        frame = getframe; %Convert figure to frame
        writeVideo(writerObj, frame); %Save frame to video file
    end

    PlotCounter = PlateCounter;

```

```

end

TimeCounter = TimeCounter + 1;
TimePassed = TimeCounter*TimeStep;
PlateCounter = PlateCounter+1;
ExitSum = sum(ExitHold,'all'); %How many cells meet the exit
requirements?

```

```

end

```

Plate0End2End.m

```

%% Initial Plate Temperature (End to End)

%% Plate 1

%% Creating Plate matrix

NumberOfColumns = (XAxisLength/GridSpacing)+1; %Calculates the number of
columns from plate x-length and grid spacing
NumberOfRows = (YAxisLength/GridSpacing)+1; %calculate the number of
columns from plate y-length and grid spacing

Plate = zeros(NumberOfRows,NumberOfColumns); %Initialise a matrix of zeros

%% Create Matrix of Temperatures at T=0

Plate(:,1) = TempL; %Define plate temperature at one end
Plate(:,NumberOfColumns) = TempR; %Define plate temperature at other end

TempStep = (TempL-TempR)/(NumberOfColumns-1); %The temperature step accross
the north and south face from one column to the next.

counter = 1; %Reset Counter

while counter<NumberOfColumns %Count through all the columns
    Plate(1,(counter+1)) = TempL - (TempStep*counter); %Calculating
temperature of all cells in the first row
    Plate(NumberOfRows,(counter+1)) = TempL - (TempStep*counter);
%Calculating temperature of all cells in the last row

    counter = counter + 1; %Count to the next column
end

Plate0 = Plate; %The previously calculated plate is the plate at time 0

%% Update temperatures at T=dt

TempCheck = 0;
PlateCounter = 1;
ExitHold = zeros(NumberOfRows,NumberOfColumns);
ExitHold(1,:) = 1;
ExitHold(NumberOfRows,:) = 1;
ExitHold(:,1) = 1;
ExitHold(:,NumberOfColumns) = 1;
ExitCriteria = NumberOfRows * NumberOfColumns;

```

```

ExitSum = 0;
PlotCounter = 1;
TimeCounter = 0;

while ExitSum < ExitCriteria

    OldPlate = Plate; %Saves plate from the previous time step

    ColumnCounter=2;

    while ColumnCounter<NumberOfColumns
        RowCounter =2;
        while RowCounter<NumberOfRows
            Tp = Plate(RowCounter,ColumnCounter); %Saves the previous value
of a cell

            Plate(RowCounter,ColumnCounter)=Fourier0*(OldPlate(RowCounter,ColumnCounter
+1)+OldPlate(RowCounter,ColumnCounter-
1)+OldPlate(RowCounter+1,ColumnCounter)+OldPlate(RowCounter-
1,ColumnCounter)-(4-(1/Fourier0))*Tp); %Calculates new value in this cell

            ExitCheck = Tp/Plate(RowCounter,ColumnCounter); %How much has
the value changed?

            %           if ExitCheck > 0.9999999 %Does the change meet the exit
criteria
                if ExitCheck > 0.999 %Does the change meet the exit criteria
                    ExitHold(RowCounter,ColumnCounter) = 1;
                elseif
                    Plate(RowCounter,ColumnCounter)==Plate(RowCounter,ColumnCounter+1)
                    %           ExitHold(RowCounter,ColumnCounter) = 1;
                end

                RowCounter = RowCounter + 1;

            end
            ColumnCounter = ColumnCounter + 1;
        end
        %           PlateName = strcat('Plate',num2str(PlateCounter));
        %           variable.(PlateName) = Plate ;

        XAxis = zeros(NumberOfColumns,1);
        counter = 1;

        while counter<=NumberOfColumns
            XAxis(counter,1) = counter;
            counter = counter+1;
        end

        YAxis = zeros(1,NumberOfRows);
        counter = 1;

        while counter <=NumberOfRows
            YAxis(1,counter) = counter;
            counter = counter+1;
        end

        %% Plot Figure

```

```

    if PlateCounter > PlotCounter + 10 || PlotCounter == 1 %Output the first
and subsequently every 10 plates

```

```

    Plot

```

```

    if VideoCheck == 1 %Does the user want video output?
        frame = getframe; %Convert figure to frame
        writeVideo(writerObj, frame); %Save frame to video file
    end

```

```

    PlotCounter = PlateCounter;

```

```

end

```

```

    TimeCounter = TimeCounter + 1;
    TimePassed = TimeCounter*TimeStep;
    PlateCounter = PlateCounter+1;
    ExitSum = sum(ExitHold, 'all'); %How many cells on the plate meet the
exit criteria?

```

```

end

```

Plot.m

```

%% Plot Figure

```

```

figure (1)
%     image(Plate)
contourf(flipud(Plate)) %creates the graph
colorbar %creates the bar on the right
colormap(jet) %sets the colour scheme for the graph

```

EndRecording.m

```

close(writerObj); %Closing the file

```

Other

$$f(x) = f(x_0) + f'(x_0) \frac{(x - x_0)^1}{1!} + f''(x_0) \frac{(x - x_0)^2}{2!} + \dots + f^n(x_0) \frac{(x - x_0)^n}{n!}$$

Equation 6: Taylor Series

$$T_p^{t+\Delta t} = F_0(T_E + T_W + T_N + T_S - \left(4 - \frac{1}{F_0}\right)T_p)$$

- $T_p^{t+\Delta t}$ = Temperature of a given node at a time Δt after time t
- T_p = Temperature of a given node at the previous time step
- F_0 = Fourrier Number = $\frac{\alpha \Delta t}{h^2}$
- T_x = Temperature of the cell in a given direction (x) at the previous time step

Equation 7: Temperature at given time step

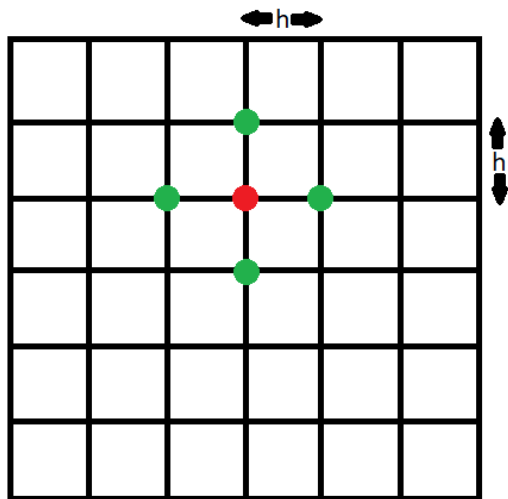


Figure 6: example of nodes on a grid